Appendix O

Title:
Platform Concept:
SMIF Specification
Version 1.0

0. Abbreviations

PP	Platform peripheral
ADCB	Address/data/control bus
Word	32 bits
Halfword	16 bits
Byte	8 bits
STA	Static timing analysis
EBI	External bus interface
BPI	Platform bus/peripheral interface

1. Introduction

Please note that all following information applies to fully digital, synthesizable Platform peripherals. Even in analog modules there is at least one digital block containing the SFR file, bus interface and control logic. The information given here is also valid for these digital parts.
Fig. 1.1 shows the PP development flow.
All PPs are first described by a C-model coming from Product Definition, and a configurable version of a synthesizable VHDL model. We call this configurable version "Softmacro", because it offers the possibility to select among several bus models where this peripheral shall be connected to, and configure the SFR address space.

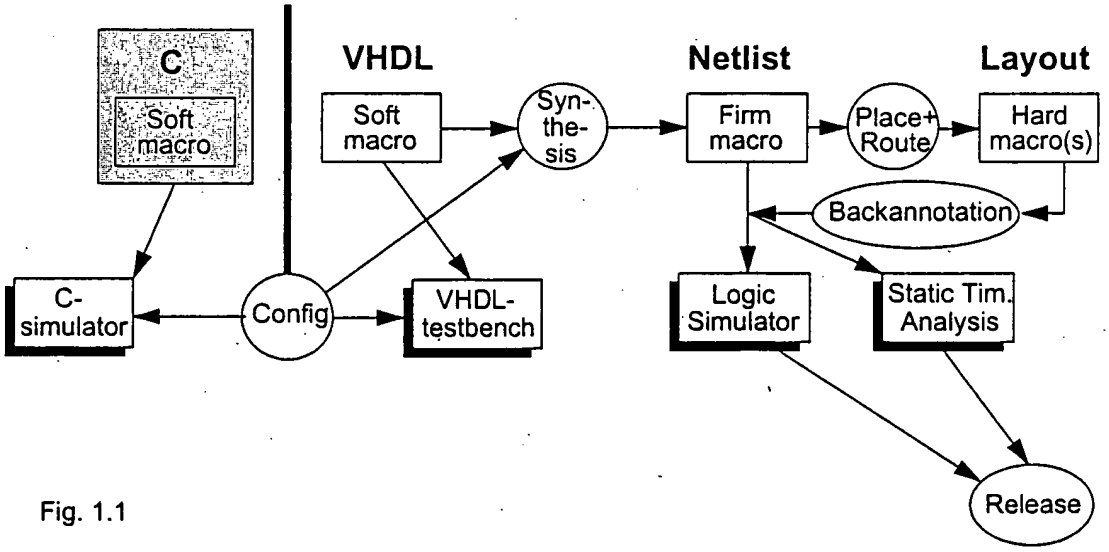

Fig. 1.1

To simulate the peripheral's behavior, the softmacro has to be configured:
one bus model has to be selected,
the SFR configuration has to be defined.
As soon as the configuration parameters have been filled out, the C/VHDL description is updated by a script. The updated C/VHDL file can be used for running functional simulations controlled by a C/VHDL testbench. The testbench is able to simulate the module's functionality by means of connecting the selected bus model.
Additionally, it can be synthesized into a firmmacro. The firmmacro is a netlist view, containing the gate-level representation of the PP. The firmmacro is simulated using a netlist-based sim-

ulator. Whereas the functional characterization took place on the softmacro C/VHDL base, the timing verification is performed on the firmmacro by using STA and backannotation (pre- and post-layout). When the timing characterization results meet the specification, the PP is released into the Platform library.

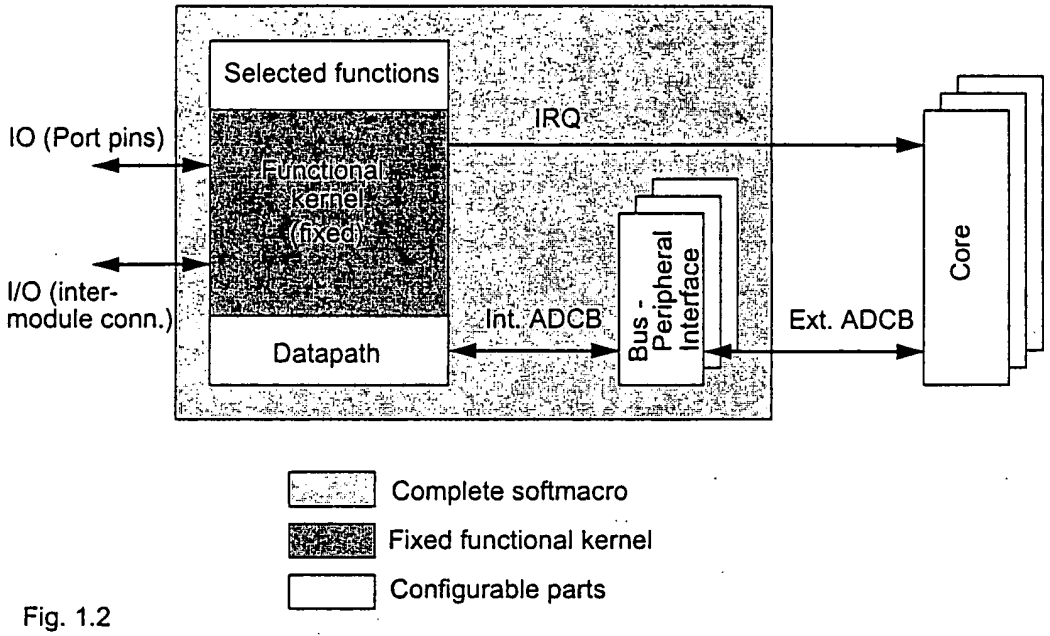

Fig. 1.2

2. Softmacro parts

Fig. 1.2 shows the view of a Platform peripheral.
Dark shaded is the fixed functional kernel which is not intended to be changed during the peripheral's lifetime. White fields indicate configurable parts of the peripheral. These are:

- additional/selected function blocks,
- SFR datapath,
- bus bridge to core.

The *light shaded* area indicates the complete softmacro. The C/VHDL files are configured by generics and constants set in packages.
The VHDL files can be synthesized into a firmmacro. The firmmacro is a netlist view, containing the gate-level representation of the PP. The firmmacro is simulated using a netlist-based simulator. Whereas the functional characterization took place on the softmacro C/VHDL base, the timing verification is performed on the firmmacro by using STA and backannotation (pre- and post-layout).

2.1. Additional/selected function blocks

They may or may not be needed in the peripheral.
Example of an additional function is a second serial communications channel. In this case, the channel's function is only described once, and if desired in the PP, it is doubled by a C/VHDL

configuration parameter.
Example of a selected function is having a timer overflow signal either high-active or low-active. In this case both versions are part of the fixed functional kernel, and the selection is made by connecting a control input signal to either VDD or VSS („Hardware switch").
Both the additional and the selected functions are configured by one or more configuration parameters which are part of the C/VHDL softmacro.

2.2. The complete datapath containing all SFR registers

The datapath width is selected automatically from the desired databus. If FPI32 is selected, all SFRs have 32 bits. For lower-performance peripherals a 16bit-wide FPI16 is defined. It is actually a part of FPI32 which is seen by a peripheral with halfword SFRs. With the Platform definition of supported busses (see section 4.2.), it is possible to:

use byte-oriented cores with byte-oriented peripherals only,
use halfword-oriented cores with halfword-oriented peripherals only,
use word-oriented cores with halfword-oriented peripherals,
use word-oriented cores with word-oriented peripherals.

2.3. Bus - Peripheral Interface (BPI)

This is the interface between the external address/data/control bus (Ext. ADCB) and the internal ADCB. Although the external data and address bus may be tristate, the internal bus system must not be tristate. Therefore separate input and output data busses are generated inside the peripherals. This is a prerequisite for performing true STA.

3. Module interface

As can be seen in Fig. 1.2, we have several groups of I/O pins which connect the peripheral to its environment:
a) I/O from/to port pins (special product-related I/O functions),
b) I/O from/to other on-chip peripherals or processor core,
c) I/O from/to on-chip ADCB,
d) interrupt request outputs.

3.1. Structural test coverage

With respect to the Platform's "Design for Test (DFT)" concept, all digital Platform peripherals will contain one or more *full scan chains* to achieve a close-to 100% structural test coverage on silicon. The required test patterns are generated on chip level via ATPG. There will be as many parallel short scan chains as can be accepted due to the number of product port pins. Every scan chain occupies two port pins (scan_in, scan_out). Scan tests are controlled by a centralized Platform module containing a JTAG-like TAP-controller. It is unique for all Platform products. For more details on the TAP controller refer to the document „dft.fm5".

3.2. Functional test coverage

During peripheral development and for the silicon's AC verification, *functional patterns* have to be provided to check the peripheral's behavior. To be independent of the product's set of peripherals, all functional tests are strictly peripheral-oriented and stimulate directly the peripheral's interface input signals. This strategy is known as *module isolation*. Only one PP can be in „module isolation" testmode at one time. If one PP is in module isolation testmode, all other PPs and the core must stay in „quiet mode", i.e. must release the ADCB and must not draw any static current. PP-related IDDQ tests are also performed using module isolation. It is therefore assumed that on product level, all the peripheral's I/O pins can be accessed directly from external with the exception that a pad input/output driver is interconnected.
This assumption is of course valid for all "I/Os from/to port pins".
All inputs coming from other on-chip modules have to be multiplexed on additional port pins.
The ADCB is expected to be accessible via port pins, too (EBI assumed). The existence of an EBI is true for most of the existing (non-Platform) products. In the age of bigger on-chip memories, more and more future products will lack an EBI.
An additional exception arises with DPI since the external DPI (visible at the product pins) runs with reduced speed compared to the on-chip (internal) DPI. The peripheral itself will be connected to the internal DPI. Synchronization is maintained by the external bus controller (EBC). Additionally the number of externally visible address lines and control signals is reduced. An EBC model is required to be part of the C/VHDL testbench in order to translate the PDL patterns into external DPI bus signals.
All outputs which are not directly accessible, are connected to a special test circuit called MISR (multiple input shift register). This MISR generates a unique signature value during the functional tests. By reading the signature after test completion, one can tell if the tests passed or failed. It is even possible to output the MISR output signal during the test. This allows error detection earlier and simplifies error tracking.
However, these DFT-related guidelines can be found in the "Design for Test (DFT)" chapter.
Platform restricts interrupt behavior to a common strategy. All Platform peripherals output their interrupt request signals. An IRQ is indicated by a HIGH pulse which lasts at least one clock cycle. The IRQ flipflops which are set by those pulses are located in an interrupt controller or (like C166 and Dolphin) in several service request nodes (SRN).

4. Configuration parameters

All configuration parameters have to be filled out by the peripheral designer. Complete set of parameters allows synthesis into a firm macro.

4.1. SFR configuration

Configuration of the special function register file is controlled by a list of SFRs, containing SFR name, address, physically available bits (e.g. [31:0], [31:16,7:0]).
Bus and datapath width:
SFRs have by default the same data width as the bus bridge width. The bus bridge interfaces the PP's SFRs to the inter-module ADCB.
Not all of the bits need to be physically available, i.e. readable or writable.
The SFR address given in the configuration list is always a byte address.
In a 16bit-peripheral SFR addresses have the least significant bit=0. Two valid-byte signals allow lower and upper byte access on a halfword-wide SFR. Halfword access is restricted to addresses with least significant bit=0.
In a 32bit-peripheral SFR addresses have the least two significant bits=00. Four valid-byte signals allow byte access and halfword access on a word-wide SFR. Byte access can take place for any of the four bytes, halfword access can take place for lower and upper halfword of the

word. Word access is restricted to addresses with least significant two bits=00; halfword access is restricted to addresses with least significant bit=0.

4.2. Bus configuration

The Bus - Peripheral Interface (BPI) is specific for the target bus the PP is intended to be connected to. The softmacro does not imply a specific BPI. For simulation with the C/VHDL testbench, the target bus has to be selected. BPI serves as an interface between the PP's SFRs and other on-chip modules which are connected to the same ADCB.

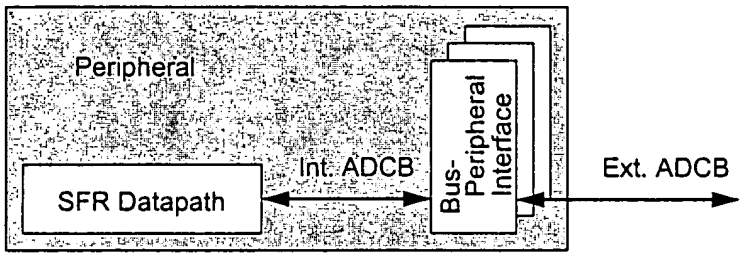

Fig. 4.2

4.2.1. External ADCB signals

Basically every ADCB supported by Platform contains the following signals. „ADCB" is replaced by the bus identifier. For final names refer to chapter 5 of this document.

	ADCB_D[n-1:0]	Data bus (I/O), width n;
	ADCB_A[m-1:0]	Address bus (I), width m;
	ADCB_WR	Write SFR (I), active high;
or	ADCB_WR_N	Write SFR (I), active low;
	ADCB_RD	Read SFR (I), active high;
or	ADCB_RD_N	Read SFR (I), active low;

Address/data bus may be multiplexed. In this case an address latch enable (ALE) signal is required.

Optional additional control signals with not predefined names serve as:

Peripheral select (I);
Ready (I/O), wait-state support;
Alive (I/O), indicates that PP address is met;
Opcode (I), select data transfer width and protocol.

The set of signals for every Platform-supported bus is listed in chapter 5 of this document.

4.2.2. Internal ADCB signals

Whereas the set of external control signals differs among the ADCBs, the internal SFR control signals are common independent of the target ADCB. The internal control signals are listed below. Note that the direction is given from view of the bus interface block.

Mandatory signals:

BPI_DATA_I[n-1:0]	Input data bus (I), width n;
BPI_DATA_O[n-1:0]	Output data bus (O), width n;
BPI_WR_SFRx_N	Write data to SFR number x (O), active low;
BPI_RD_SFRx_N	Read data from SFR number x (O), active low;

Optional signals:

BPI_WR_BY_N[k:0]	Select SFR bytes for write operation (O), active low, k=1 for 16bit PP, k=3 for 32bit PP;
BPI_RDWR_N	Read-modify-write indication (O), active low.
PER_RDY	Ready signal of peripheral kernel

There are two reasons for separating the data input and output busses:
1) a tristate bus complicates static timing analysis, ATPG, synthesis, etc.
2) since a read operation takes place in cycle #n, and a write operation takes place in cycle #n+1, bus contention would appear if read and write data shared the same bus; see fig. 4.2.2.

Fig. 4.2.2.1 shows a sequence of „read SFR data #1 / write SFR data #2 / read SFR data #3". As an example the signal timing is taken from PPI bus:
Cycle #1 rising CLK drives address for read data #1 on external bus ADCB_A and activates internal read signal BPI_RD_SFRx_N; the read SFR data #1 is expected to appear on internal data bus BPI_DATA_I before cycle #2 rising CLK.
Cycle #2 rising CLK drives address for write data #2 on external bus ADCB_A, and drives read data #1 on external bus ADCB_D. Simultaneously, the write data #2 is announced to the bus interface, but is actually driven one cycle later (DPI bus pipelining).
Cycle #3 rising CLK address for read data #3 on external bus ADCB_A, and drives write data #2 on external bus ADCB_D. This data is immediately propagated onto the internal data bus BPI_DATA_O, and write signal BPI_WR_SFRx_N is active. Simultaneously, internal read signal BPI_RD_SFRx_N is activated; the read SFR data #3 is expected to appear on internal data bus BPI_DATA_I before cycle #4 rising CLK.
In cycle #3 internal read and write signals are simultaneously active, and read and write data are driven simultaneously on the internal I- and O-busses, respectively.

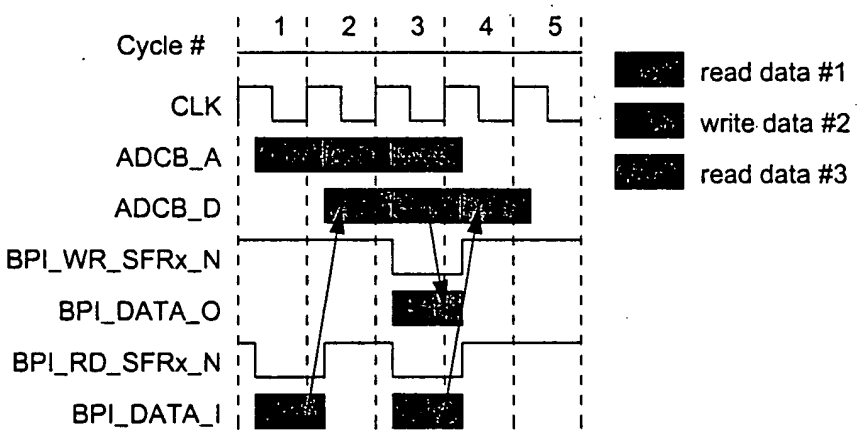

Fig. 4.2.2.1

Fig. 4.2.2.2 shows a sample BPI for an external Platform bus interfacing to a passive slave PP. Note that two additional signals connect BPI and peripheral kernel. Signal BPI_RMODW_N indicates to the peripheral kernel that a read-modify-write access is in progress. Signal

PER_RDY is needed if a peripheral is slower than the bus. This signal indicates when the peripheral has finished with the access.

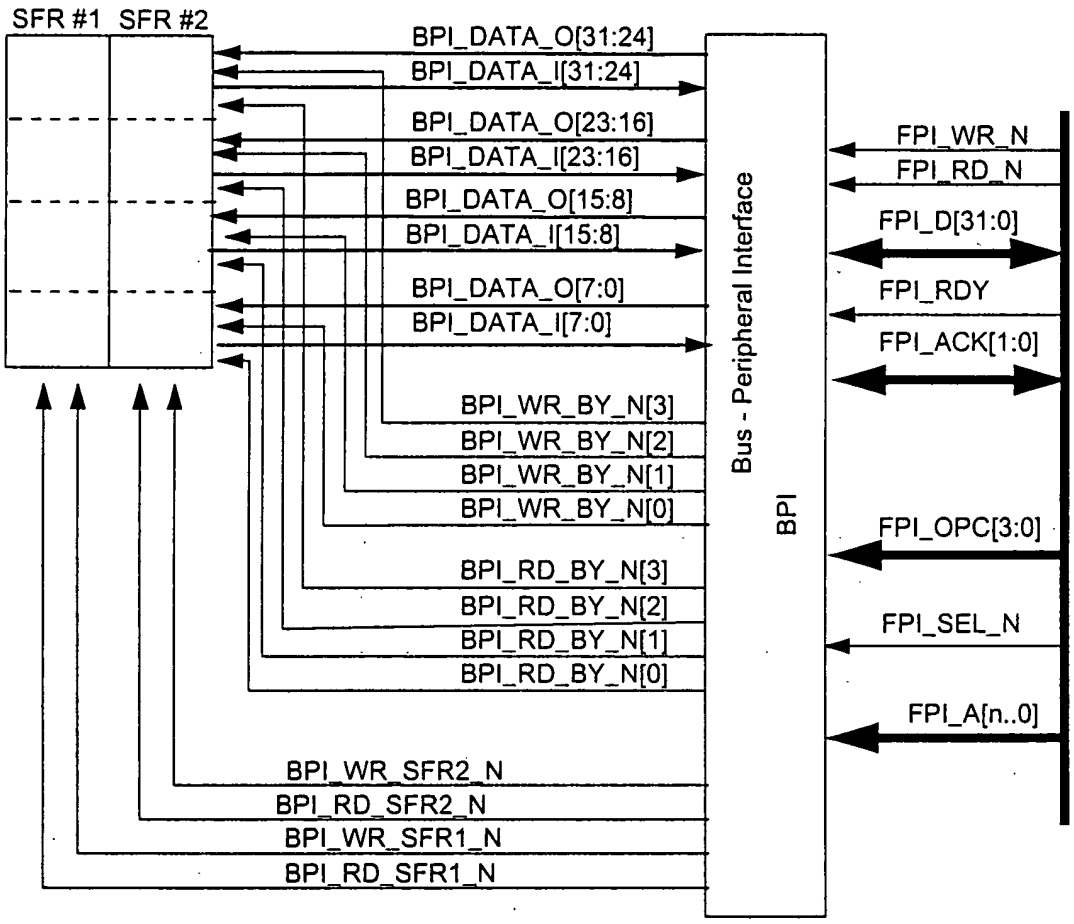

Fig. 4.2.2.2

4.2.3. SFR data widths

Not necessarily, the ABCD bus width (i.e., the width of the master) is the same as the width of a passive slave PP. This gives us the following scenarios:

Tabelle 1:

ADCB data width	SFR register width	SFR access width	Comment
32	32	32	word access
32	32	16	half word access
32	32	8	byte access
32	16	16	word or half word access
32	16	8	byte access
32	8	8	any width for access
16	32	16	sel.halfword by addr.
16	16	16	standard
8	32	8	sel. byte by addr.
8	16	8	sel. byte by addr.
8	8	8	standard

For a 32 bit bus, each SFR address must be mapped to a 32 bit address regardless whether the SFR width is 8, 16, or 32 bits. (This means for 32FPI, that each SFR address ends on 00 even if the SFR is only 8 bit wide.) Similarly for a 16 bit bus, each SFR address of an SFR with width 8 or 16 bit must be mapped to a 16 bit address.
Data is shown on the bus at the location where it resides in the SFR. If for example the second byte of a 32 bit register is read, this byte is visible on the second byte of the bus as well.
To make the scenarios clearer, let us pick examples for the FPI bus:

If the 32bit SFR shall be written to an 8bit ADCB, the data byte has to be directed to its proper target location. This is done by the internal control lines BPI_WR_BY_N[3:0]. They have the same timing as the write signals BPI_WR_SFRx_N. Note that a problem will arise here if read a continuously changing 32 bit SFR (e.g., a 32 bit counter value). Since such a value will have changed until the next byte is read, we have to think of implementing a 32 bit buffer register that stores the value when the first byte is read. And when the next byte is read in a later cycle, it is read from this buffer register.
Similar strategies apply to a 16bit PP connected to an 8bit ADCB and to a 32bit PP connected to a 16bit ADCB. The byte and halfword data are expected to appear on their corresponding positions on the external ADCB. Fig. 4.2.3 shows the mapping of SFR word/halfword/byte addresses to the internal BPI_WR_BY_N[3:0] lines.

For read access, usually the complete 32 bits are read even if only a byte access occurred. This is the easiest way but may cause trouble for destructive read (an SFR changes its value as soon as it is read). For this case, also signals BPI_RD_BY_N[3:0] are provided by BPI and can be used by the peripheral kernel optionally.

PPs with more SFR bits can be connected to ADCBs with less data bits; e.g. a 32bit timer can be connected to PB8. In this case, since only one byte can be accessed at one time, the Platform standard says:
A 32bit SFR is built from 4 consecutive bytes. The basic SFR address is ...xx00. This is also

the address of its least significant byte. The SFR contains also bytes ...xx01, ...xx10 and ...xx11. Byte ...xx11 is the most significant byte.
A read access on SFR byte ...xx00 loads the complete 32bit value of the SFR into a buffer. The other 3 bytes have to be read from this buffer consecutively by applying addresses ...xx01, ...xx10 and ...xx11 in any order. *Important is that any access to byte ...yy00 of another SFR will immediately overwrite the current buffer contents.*

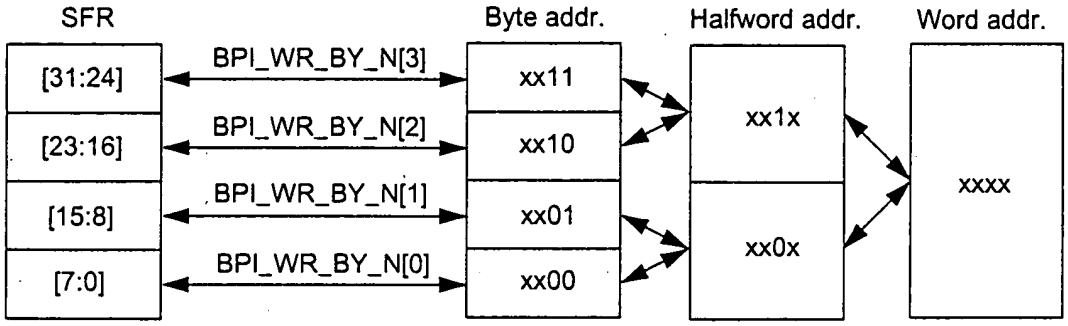

Fig. 4.2.3

Please note again that an SFR read access does always read the complete SFR, even if FPI_OPC[3:0] denotes a halfword or byte access. Halfword and byte access is only evaluated for write accesses.

4.3. Functional configuration

This feature to add or select functionality is optional. If some functional subblocks shall be omitted or exchanged, VHDL models of each subblock must be provided. According to the value of the functional parameters, the corresponding subblocks' VHDL models are merged into the softmacro description.
However, the functional kernel of a Platform peripheral will never be changed. Functional configuration parameters are needed only to add functionality to the kernel or enable/disable certain functions within the kernel.

4.4. BPI architecture

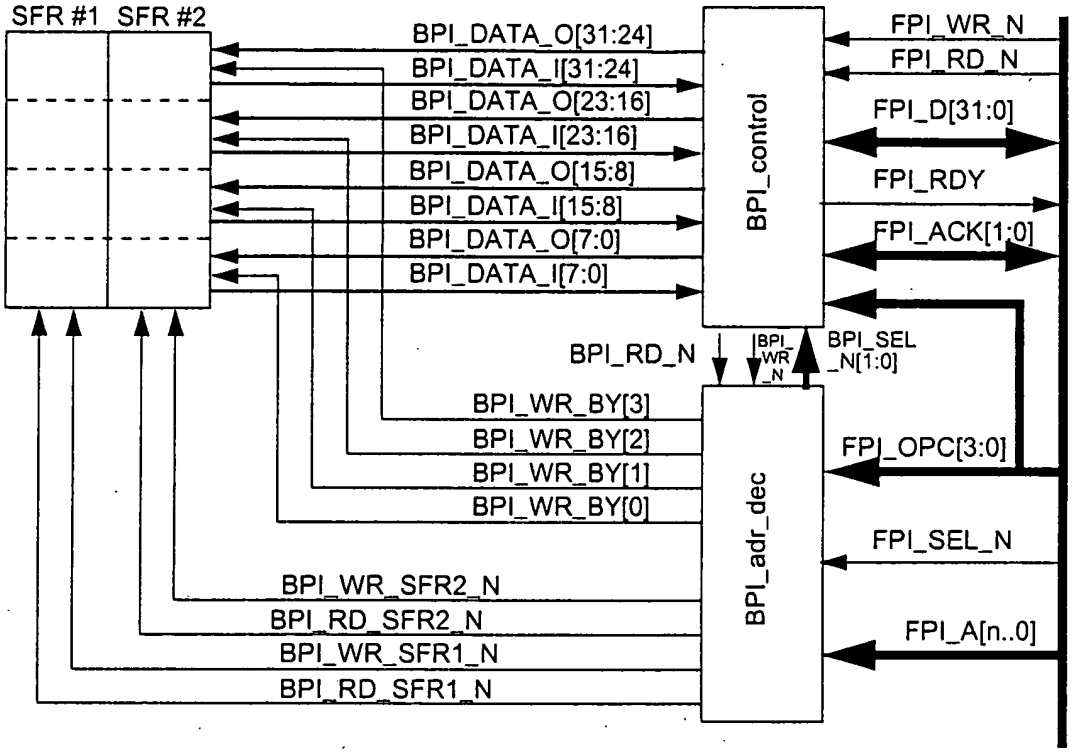

Fig. 4.4.

Fig. 4.4. illustrates the BPI architecture for the example of an external Platform bus interfacing to a passive slave PP. Signals BPI_RD_BY[3:0] are omitted in Fig. 4.4 for sake of simplicity. Basically, BPI consists of two parts, the BPI control unit (BPI_control) and the BPI address decode unit (BPI_adr_dec).

BPI_control connects external and internal data busses, passes read/write information to BPI_adr_dec, and drives other control information such as RDY (ready) and ACK (acknowlegde).

BPI_adr_dec decodes the external address bus and evaluates which SFR is accessed. It ands this information with the read/write information from BPI_control and drives the SFR-specific read/write lines (BPI_RD_SFRx_N, BPI_WR_SFRx_N). Simultaneously, BPI_adr_dec generates the byte-select signals for writing byte- or halfword-parts of a SFR from either FPI_OPC[3:0] or it may be hard coded that, e.g., only 16 bits accesses are possible if the external bus has 16 bit. In case of FPI, a pre-decoded select signal called FPI_SEL_N is provided by the central bus controller.

In case of a byte (or 16 bit) access to a wider SFR (32 bit SFR), there are two scenarios. Firstly, if a 32 bit bus is connected to BPI, the whole data bus is forwarded by the BPI_control. In case of a read access, the complete SFR is read. In case of a write access, the complete bus is forwarded to the internal bus and the peripheral kernel has to activate only the write lines of the selected bits. Secondly, if a bus smaller than 32 bits is connected. It must be provided that the selected byte of the SFR (e.g., the highest byte of the SFR) is shifted to a lower location so that it can be read by the external bus. The opposite shift operation is needed for write access.

5. Platform-supported ADCBs

This chapter describes all ADCBs which are supported by Platform. A complete list of signals is given, plus a connection block diagram and a timing diagram.

5.1. 8-bit peripheral busses

5.1.1. PB

PB_AD[7:0]	Address/data multiplexed, address range 00h..FFh (Note: 8051 architecture reserves addresses 00h..7Fh for on-chip RAM).
PB_ALE	Address latch enable, active high (was originally LDIRA_B),
PB_RD_N	Read SFR data, active low (was originally RDIRA_B),
PB_WR_N	Write SFR data, active low (was originally WDIRA_B).

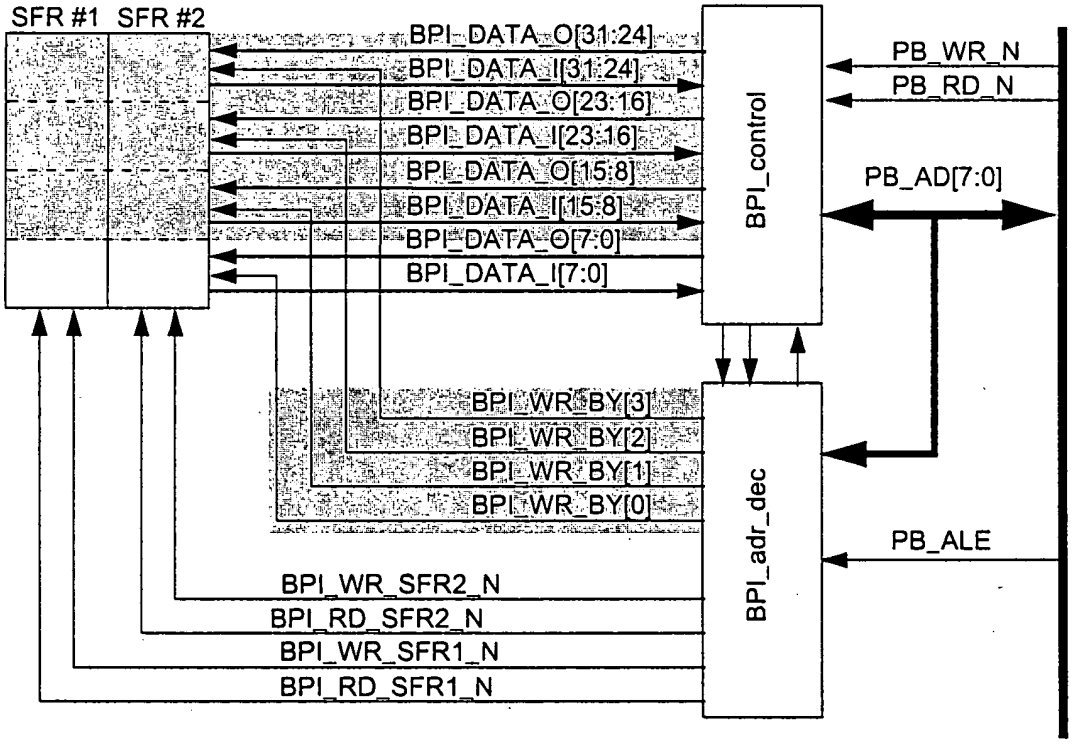

Fig. 5.1.1.1. mean optional architectural parts

Figure 5.1.1.2 illustrates read and write access. Let SRF1 have address adr1 and SFR2 adr2.

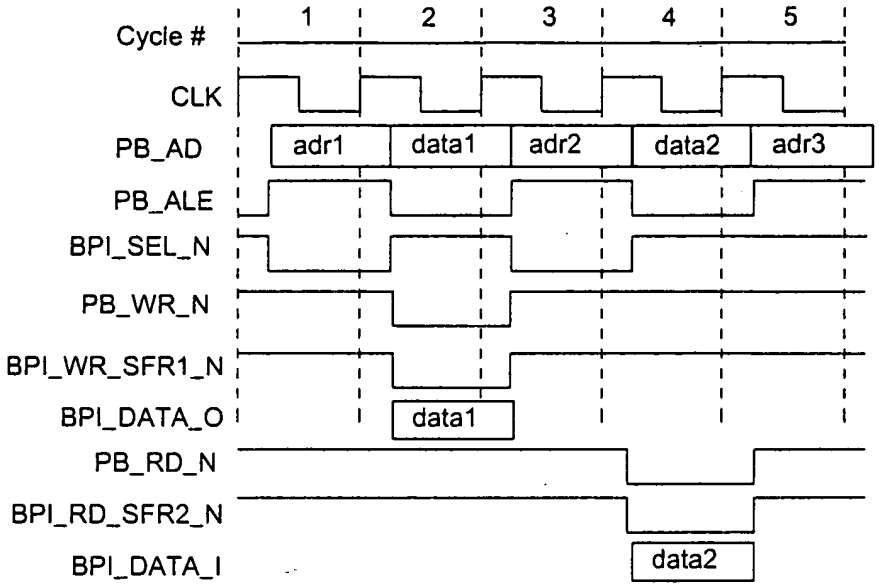

Fig. 5.1.1.2

BPI_adr_dec decodes PB_AD during PB_ALE. With rising clock edge in the beginning of cycle 2, the coming access to SFR1 is detected. With PB_WR_N going low, BPI_WR_SFR1_N is driven low so that SFR1 can be set to the value on BPI_DATA_O with the next rising clock edge. Also, BPI_adr_dec drives BPI_SEL_N active so that BPI_control drives data1 onto BPI_DATA_O. A similar scenario applies to the read access in the next two cycles. The address adr3 is not an address of any SFR in the considered peripheral so that BPI_SEL_N stays inactive.

Contrary to Figure 5.1.2, the current implementation of PB always precharges PB during CLK='1'. Thus, all reads and writes can only be active during CLK='0'. However, this is subject to change for an 8 bit platform core and thus not considered here.

5.1.2. MB

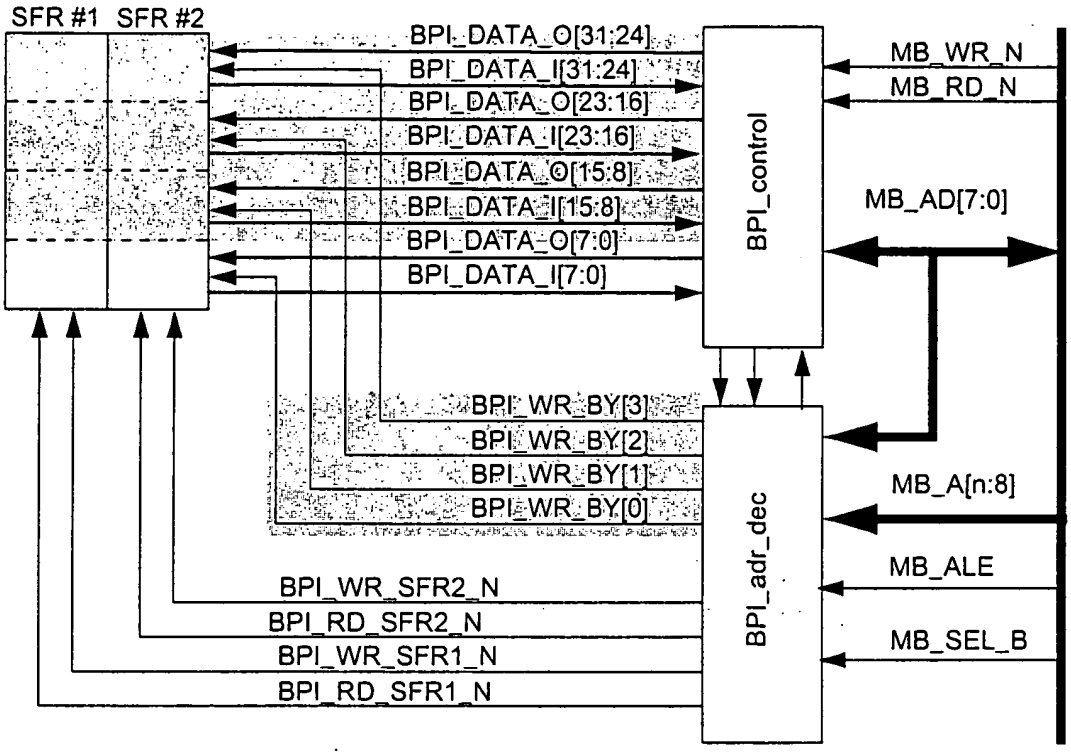

Fig. 5.1.2.1. mean optional architectural parts

MB replaces the original AMO-XDB. To avoid naming conflicts with the 16bit XBUS (=XB in Platform), XDB has been renamed to MB (Memory bus).

MB_AD[7:0]	Low address/data multiplexed,
MB_A[15:8]	High address byte, address range 0000h..FFFFh Note: not all most significant address lines need to be connected.
MB_SEL_B	PP select, active low (was originally CSCODE_B or CS_XRAM_B),
MB_ALE	Address latch enable, active high (was originally XDBALE),
MB_RD_N	Read SFR data, active low (was originally RD_B),
MB_WR_N	Write SFR data, active low (was originally WR_B).

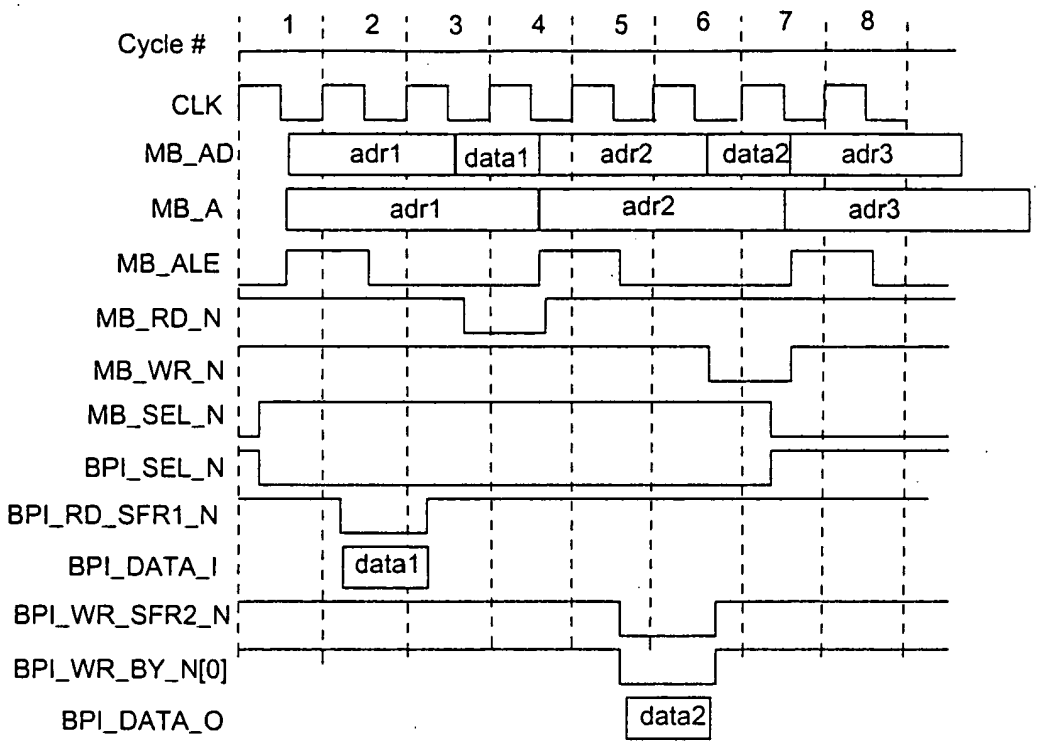

Fig. 5.1.2.2.

Figure 5.1.2.2. shows a read and a write access with MB. The exact timing is not yet specified for a new-to-develop 8 bit platform core.

5.2. 16-bit peripheral busses

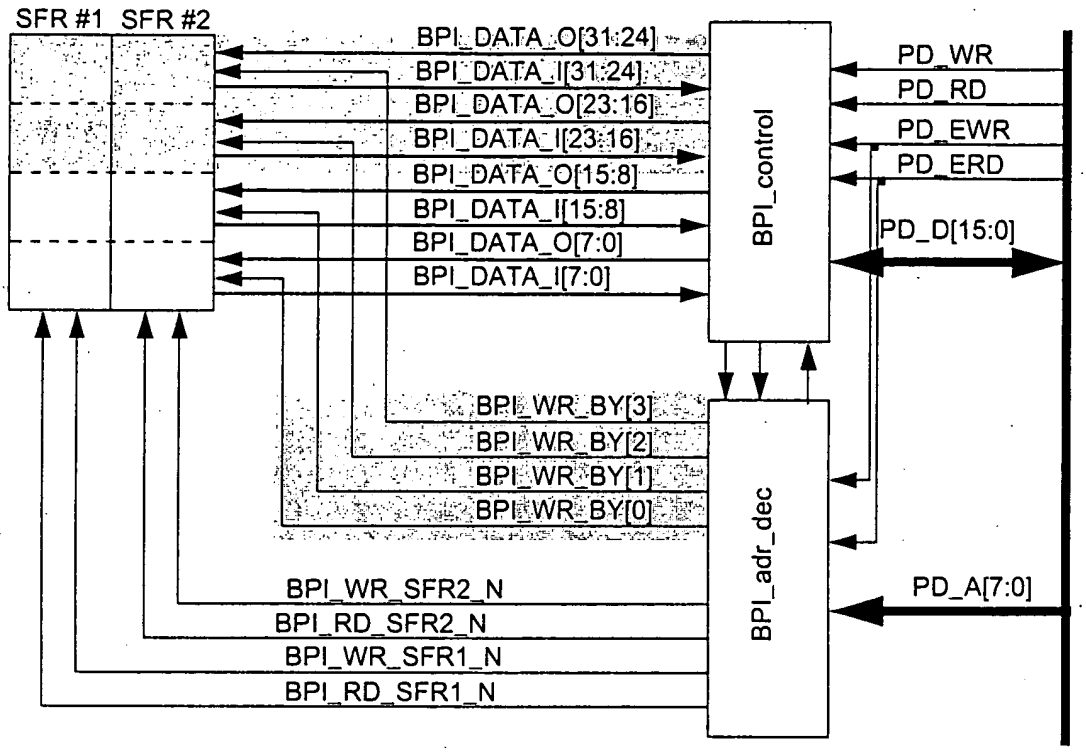

Fig. 5.2.1.1. mean optional architectural parts

5.2.1. PD

PD_D[15:0]	Data (was originally PD[15:0]),
PD_A[7:0]	Address, range 00h..FFh (was originally PA[7:0]),
PD_RD	Read SFR data, active high (was originally DW_PRD),
PD_WR	Write SFR data, active high (was originally DW_PWR),
PD_ERD	Read extended SFR data, active high (was originally DW_EPRD),
PD_EWR	Write extended SFR data, active high (was originally DW_EPWR).

Note: PD_RD/PD/WR and PD_ERD/PD_EWR select different 256 SFR blocks, thus a total of 512 SFRs are available.

Figure 5.2.1.1. shows PD_EWR and PD_ERD be connected to BPI_adr_dec. This is because extended read or write is actually an SFR extension, i.e., if PD_EWR is used instead of PD_WR, a different SFR is accessed with the same address.

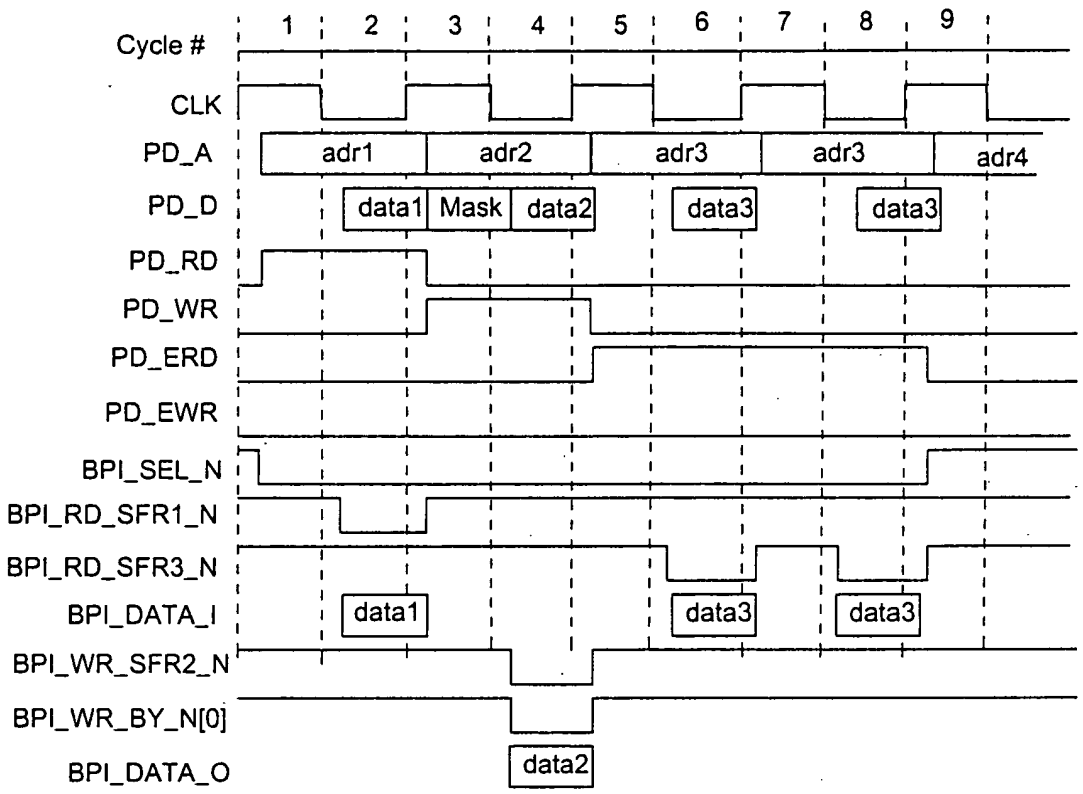

Fig. 5.2.1.2.

Figure 5.2.1.2. illustrates a normal read, a normal write access and an extended read access. Note that setup and hold times of the PD bus are defined in the PD bus specification (PD_Bus Specification V0.2, Axel Freiwald). These times have to be met in order to allow connecting BPI to the PD in an old device. Thus, VHDL description and synthesis has to meet be constrained carefully and also simulation after synthesis (and layout) has to be carried out thoroughly.

5.2.2. XB

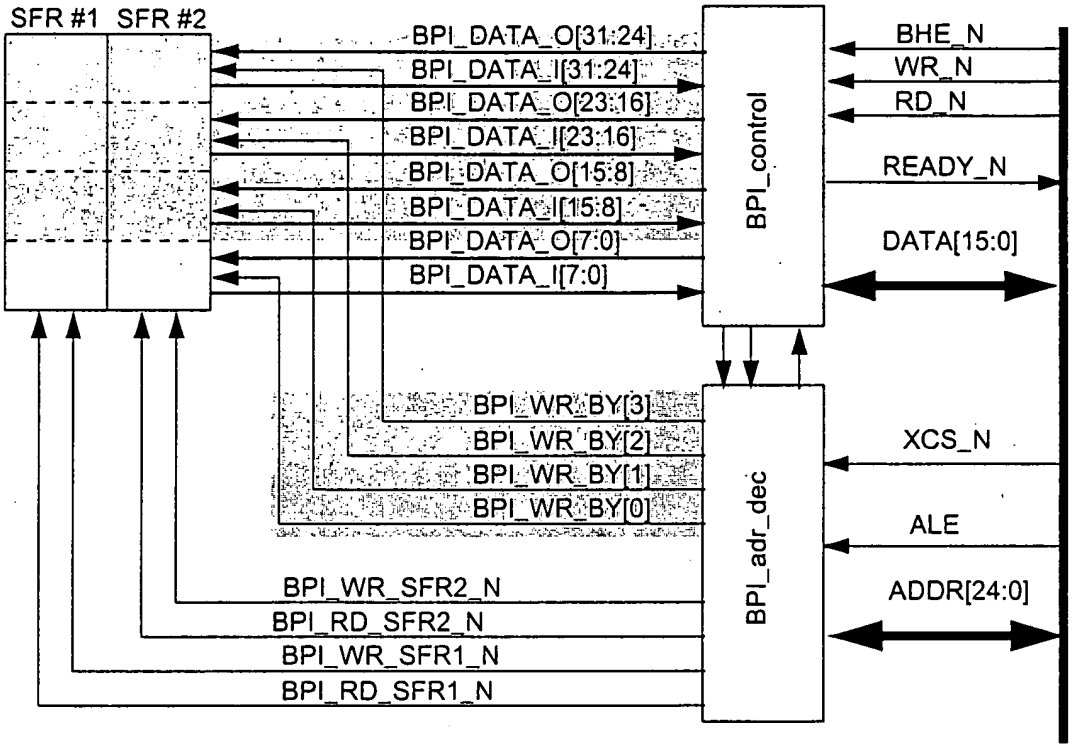

Fig. 5.2.2.1. mean optional architectural parts

XB replaces the original XBUS.

The XB is not very nice to implement in a synchronous environment due to its timing specification. This holds for the synchronous mode and is even worse for the asynchronous mode. Therefore, a BPI for XB will hardly be implemented in a strict synchronous fashion. This induces that significant problems will arise if static timing analyses, ATPG, etc. shall be applied. Nevertheless, it is feasible to develop a BPI even for the XB asynchronous mode. Thus, here are some basic ideas.

XBUS_data[15:0] data,
XBUS_addr[23:0] address, address range 000000h..FFFFFFh (16 Mbytes)
Control lines (CS_B, RD_B, WR_B, READY, RST), support of several bus protocols incl. wait states.

Synchronous mode:

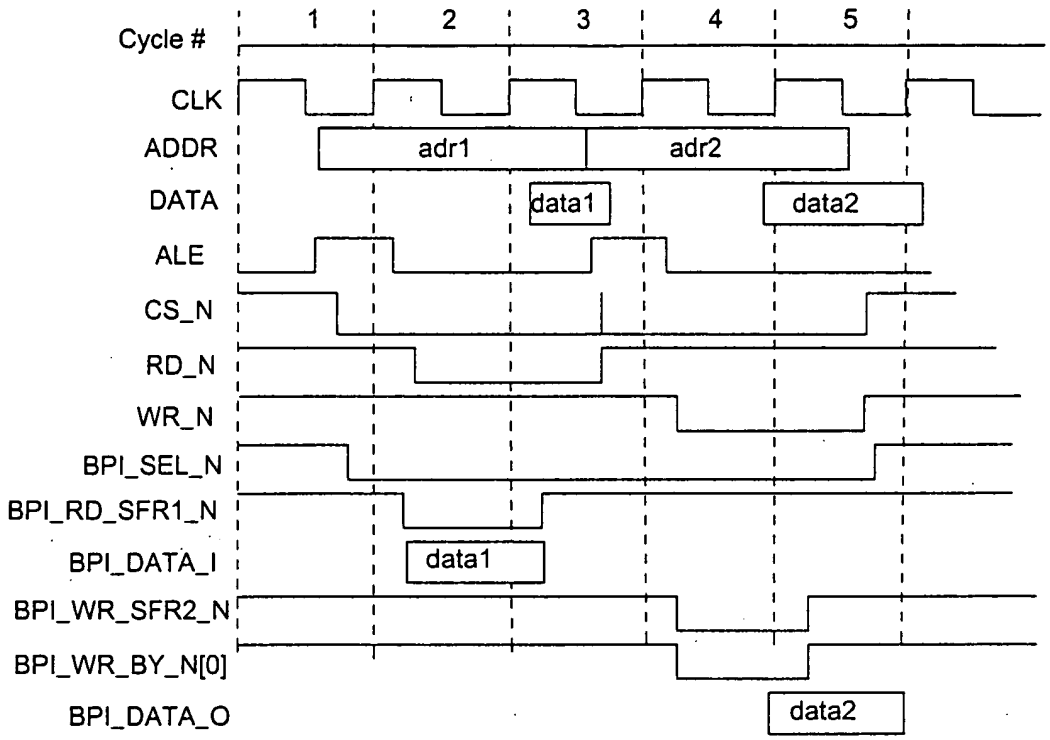

Fig. 5.2.2.2.

The general scheme for normal read and write access can be seen in Figure 5.2.2.2. Note that the timing specification of the XBUS has to be met.

Let us hint on two unpleasant features. For read access, data is never driven during a rising clock edge but around a falling edge (see data1 on signals DATA). Thus, driving data onto the DATA bus must be carried out by some not strictly synchronous logic. For write access, the data (data2 on DATA) are valid only very shortly before the rising clock edge. So, there might not be enough time to write this data to a SFR with the rising clock edge. If this is the case, data should be written with the next rising clock edge (beginning of cycle 6 in Figure 5.2.2.2.). If this is the case, some changes are necessary to Figure 5.2.2.2: Signal BPI_SEL_N must be prolonged by one clock cycle, BPI_WR_SFR2_n and BPI_WR_BY_N[3:0] are valid one clock cycle later, and data2 will be seen on DPI_DATA_I one clock cycle later.

BPI_adr_dec should store the address on ADDR with the rising clock edge if ALE is high and CS_N is low. Then the stored address has to be decoded.

Asynchronous mode:

XB can also be used in an asynchronous mode. Then, all timings are related to the falling edge of ALE. No relationship to a clock is mandatory. We may want to proceed as follows. For read access, BPI_RD_SFRx_N can be activated as soon as the address is recognized to be valid, which is at the falling edge of ALE. To drive data with the correct timing onto bus DATA, some

asynchronous logic will be necessary. For write access, BPI_WR_SFRx_N and BPI_WR_BY_N should be activated as soon as the data is available. If now read or write access needs more time because for example the XB is clocked faster than the peripheral, wait statements can be inserted by a peripheral using READY_N.

Wait state insertion:

The duration between falling edge of ALE (i.e., when the address is valid) and the time when the data is or has to be valid can be extend by one or more clock cycle using the ready signal. Also for this feature, please refer to the XBUS specification.

5.2.3. FPI

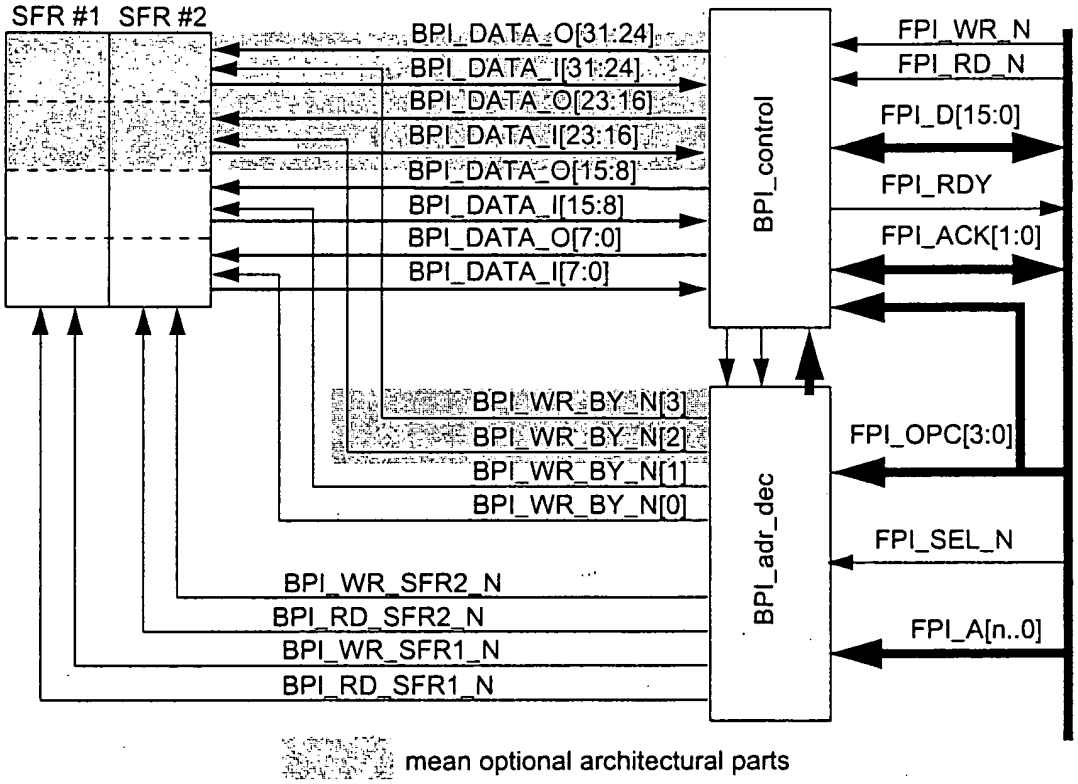

Fig. 5.2.3.1.

Mandatory signals:

FPI_D[15:0]	Data, only the lower 16 of the 32 FPI data lines are connected,
FPI_A[31:0]	Address, range 00000000h..FFFFFFFFh (4 Gbytes), not all address lines will be connected,
FPI_SEL_N	Chipselect, active low, might be common to several PPs,
FPI_RD_N	Read SFR data, active low,
FPI_WR_N	Write SFR data, active low,

FPI_RDY	Ready signal, active high, indicates waitstates,
FPI_OPC[3:0]	Opcode, indicates data width and data transfer protocol,
FPI_ACK[1:0]	Slave response code.
FPI_RDY	Ready signal from peripheral, active high, indicates waitstates if inactive

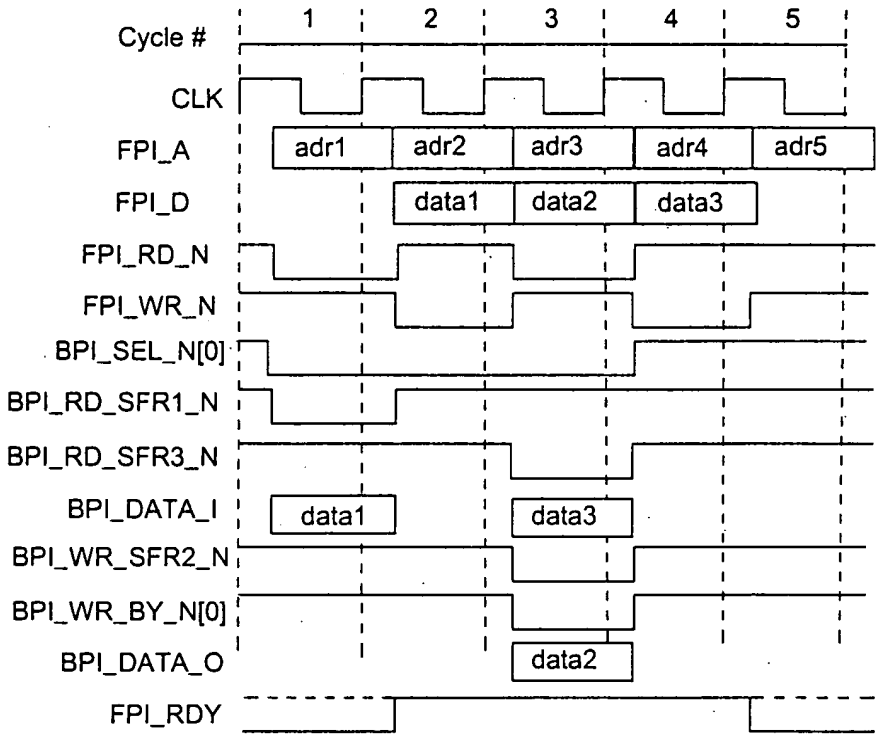

Fig. 5.2.3.2.

Figure 5.2.3.2 illustrates simple read and write accesses without wait states. FPI is pipelined so that the data that belong to an address appear one clock cycle later. For this reason, care has to be taken that read and write accesses and the data are forwarded at the right time. As indicated in Figure 5.2.3.2, write signals are delayed by one clock cycle while write data are forwarded immediately. For read access, we have a different story. The read signal is forwarded immediately so that also the read data are received by BPI immediately (immediately means in the same clock cycle). With the next rising clock edge, these data are driven onto FPI_D.

Note that an implementation problem might arise here. This is if decoding the address on FPI_A, switching BPI_RD_SFRx active, and returning the corresponding data to BPI takes more time than one clock cycle. If it turns out that this is not feasible in one clock cycle, then we must redefine BPI in a way that the peripheral kernel drives out data one clock cycle later and these data are then forwarded immediately to FPI_D by BPI.

Implementation hint: As FPI_SEL_N is decoded by a bus controller, this decoding will cause a delay on this signal. This means that the address on FPI_A is valid earlier than FPI_SEL_N. Thus, the address should be decoded first and determined valid or not with FPI_SEL_N, then.

Let us now discuss the purpose of signals BPI_SEL_N[1:0] (see also Figure 4.4, signals between BPI_control and BPI_adr_dec). Signal BPI_SEL_N[0] is the forward of signal

FPI_SEL_N. BPI_SEL_N[1] is activated if the address on FPI_A meets any SFR address of the peripheral kernel. These signals are necessary for BPI_control to be able to recognize whether a valid address is on FPI_A when FPI_SEL_N is activated. If this is the case, BPI_control must drive FPI_RDY active. Otherwise, BPI_control must drive an error message on FPI_ACK.

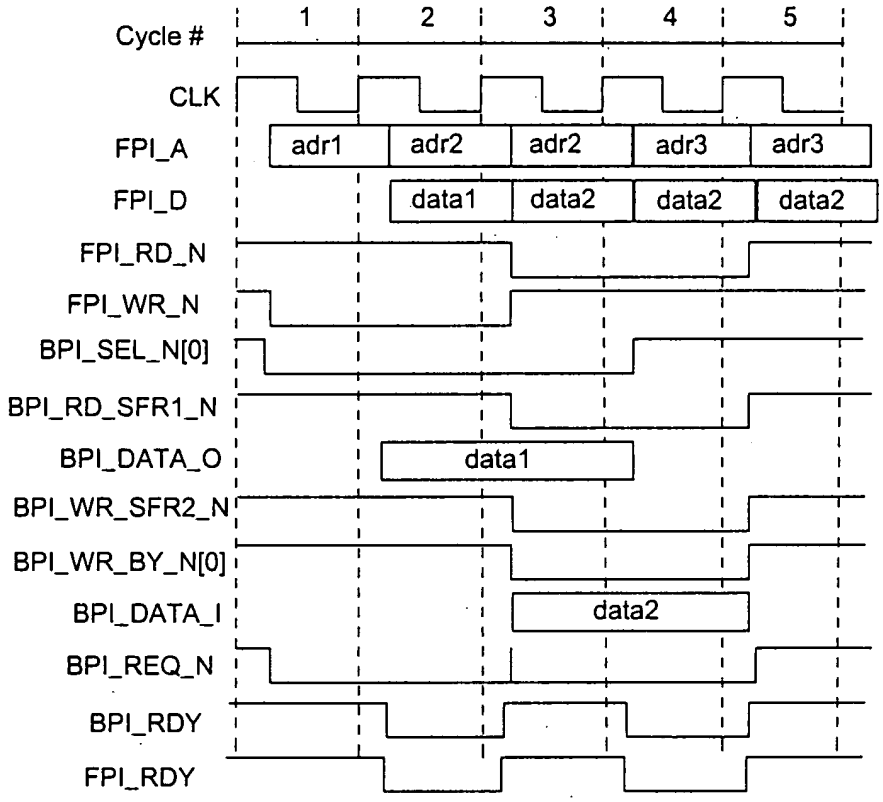

Fig. 5.2.3.3.

Figure 5.2.3.3. shows BPI connected to a peripheral that inserts one wait state for each access. This can be necessary for a slow RAM or a peripheral with a slow clock.
Similar as shown in Figure 5.2.3.2, the write signal is delayed by one clock signal. For read, the read data are delayed by one clock signal.

The number on inserted wait states is flexible. It depends on when the BPI_RDY signal from the peripheral kernel is active. This leaves the full flexibility of how many wait states must be inserted to the peripheral kernel. However, at least one wait state is required then. This restriction is necessary since we need one wait state to make RDY signal handshake.

If a write is followed by a read access, two different implementations of BPI can be selected that avoid that the old data is read. First alternative inserts an additional wait state. Second alternative provides multiplexers to forward data.

5.3. 32-bit peripheral busses

5.3.1 FPI

Signals:

FPI_D[31:0]	Data,
FPI_A[31:0]	Address, range 00000000h..FFFFFFFFh (4 Gbytes), not all address lines will be connected,
FPI_SEL_N	Chipselect, active low,
FPI_RD_N	Read SFR data, active low,
FPI_WR_N	Write SFR data, active low,
FPI_RDY	Ready signal, active high, indicates waitstates,
FPI_OPC[3:0]	Opcode, indicates data width and data transfer protocol,
FPI_ACK[1:0]	Slave response code.

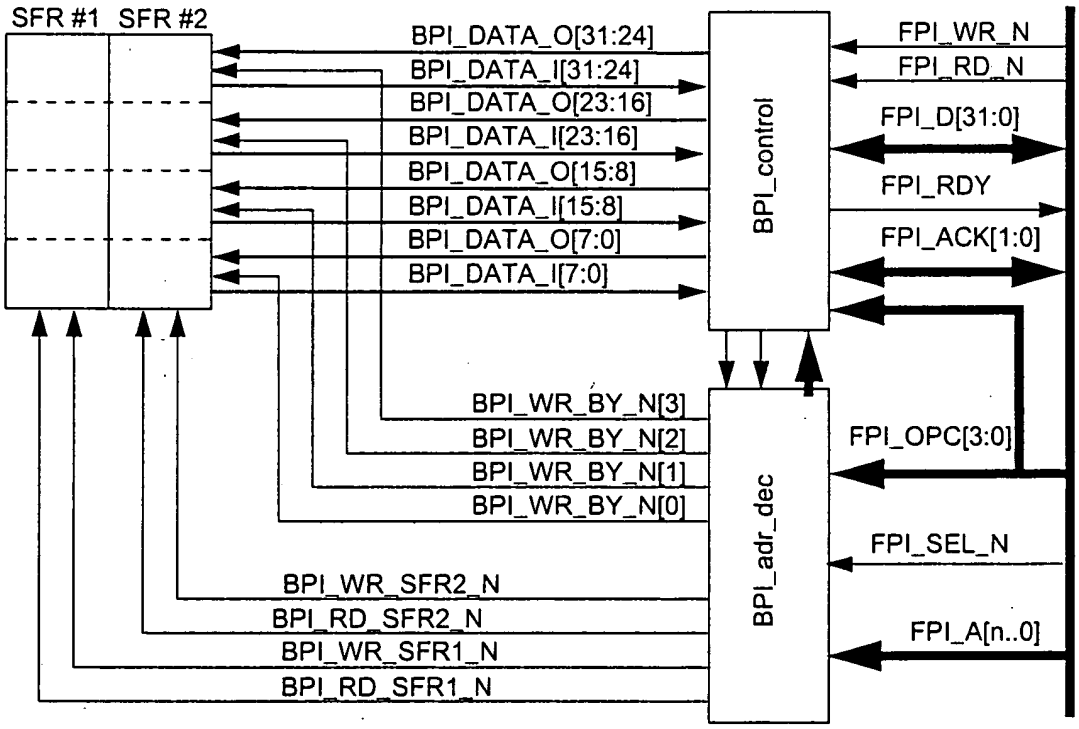

Fig. 5.3.1

Note: The number of control lines and thus the complexity of the bus interface grows with the required functionality: passive slave is easiest, intelligent master is most complex. BPI supports only passive slaves.

6. Bit Access

A bit write works as follows. The CPU performs consecutively a read and a write. For FPI bus, where a multi master concept applies, a read modify write access is performed which induces that the bus is locked. The whole read SFR is modified only in the bit that has to be changed. The whole value of the SFR is then written to the SFR.

Bit protection mode:
If a bit is bit protected, we must take special care. Let us assume a write bit access is performed by read modify write. In the affected SFR, also a bit protected bit exists. Now imagine that during the write access due to the read modify write, the hardware also wants to write to the bit protected bit. Normally, the software will win and the 'old' value will be written to the bit protected bit. This is ok for bits which are not bit protected. For bit protected bits, we proceed differently. BPI checks for each bit whether its value has changed and provides a signal that indicates whether its value has changed. This is protect_bus_o[31:0]. A line of protect_bus_o going high indicates that the value of the corresponding bit has changed. A write access is now performed only for the bit where protect_bus_o indicates a new value. Thus, if the value of a bit is not changed by software (the case during a bit write to a different bit) then the hardware can write to this bit.

7. Interrupt Register/Node Handling

To maintain the full flexibility for the implementation of the interrupt node, the interrupt nodes are not included in the peripherals. This means that the interrupt registers are not part of the peripheral module. However, BPI delivers read and write signals for the interrupt registers that are related to the current peripheral. So we get as an output of the peripheral:
BPI_WR_<interruptregister>_N, BPI_RD_<interruptregister>_N.

8. Implementation and VHDL related stuff

Please find under the platform page on the intranet "VHDL Testbench Concept" by Thomas Hillman.

Please find under the platform page on the intranet "BPI Specification Draft" by Helmut Steinbach and Andreas Weisgerber.